KATHOLIEKE
UNIVERSITEIT
LEUVEN

DEPARTEMENT TOEGEPASTE ECONOMISCHE WETENSCHAPPEN

RESEARCH REPORT 0221

AN LP-BASED LOWER BOUND FOR THE SIMPLE ASSEMBLY LINE BALANCING PROBLEM

by
M. PEETERS
Z. DEGRAEVE

D/2002/2376/21

AN LP-BASED LOWER BOUND FOR THE SIMPLE ASSEMBLY LINE BALANCING PROBLEM

Marc Peeters

Katholieke Universiteit Leuven, Department of Applied Economics, Leuven, Belgium

Zeger Degraeve

London Business School, Sussex Place, Regent's Park, London NW1 4SA, United Kingdom

Katholieke Universiteit Leuven
Department of Applied Economics
Naamsestraat 69
B-3000 Leuven
Belgium
Fax. + 32 16 32 67 32

e-mail: zdegraeve@london.edu
marc.peeters@econ.kuleuven.ac.be

April 16, 2002

Abstract:

The simple assembly line balancing problem is one of the classical integer programming problems in the operations research literature. It has many industrial applications, e.g. in the car industry. It consists of spreading the workload to make a unit of a finished product over the assembly line. More specifically, a set of tasks, which is an indivisible amount of work requiring a number of time units, must be assigned to workstations without exceeding the cycle time. Many heuristics and exact branch-and-bound algorithms are proposed to obtain solutions for the simple assembly line balancing problem. In this paper, we present a new lower bound, namely the LP relaxation of a formulation based on Dantzig-Wolfe decomposition. We assess the quality of the lower bound by comparing it with the best known solution of the various instances from the literature.

key-words: Integer programming algorithms: Dantzig-Wolfe decomposition; production/scheduling: applications

1 Introduction

An assembly line consists of a number of workstations in which tasks are performed to obtain a sequence of finished product types. A conveyer belt moves an unfinished product from one station to the next at constant speed. The time interval between the finishing of two units of the finished product type is called the cycle time. As a result, the total time that an unfinished product can spend in a station along the line can not exceed the cycle time. This total time is equal to the sum of the processing times of each task performed in the station. Some tasks can only be started after another task is finished, because of technological constraints. Those precedence relations can be represented by an acyclic precedence graph with, for each task $i \in N = \{1,...,n\}$, a node and arcs (i,j), if task i must be completed before task j. Throughout this paper, we assume that if task i precedes task j than i is smaller than j. We only consider the Simple Assembly Line, which means that only one model is made by the assembly line system.

The stations $S_1,...,S_m$ are a partition of the set of tasks N, where station s precedes station r along the assembly line, if $s<r$. The SALBP type I (SALBP-I) can be defined as follows. A set of n tasks with processing time t_i for $i = 1,...,n$ must be assigned to the minimum number of stations m such that the cycle time c for each station is not exceeded and the precedence relations between the tasks are not violated, i.e. for all arcs (i,j) of the precedence graph, it holds that $s \leq r$, if $i \in S_s$ and $j \in S_r$. The SALBP type II (SALBP-II) minimizes the cycle time c for a fixed number of stations m.

The Simple Assembly Line Balancing Problem (SALBP) is an intensively studied problem in Operation Research. Over the last four decades, many heuristics and optimal procedures have been proposed. Baybars (1986) presents a survey of the literature, focussing on exact algorithms for the SALBP-I and Talbot et al. (1986) present an evaluation of heuristic line balancing techniques. More recently, Scholl and Klein (1997) present "SALOME", a new branch-and-bound algorithm for the SALBP-I and compare it with the algorithms "EUREKA" of Hoffmann (1992) and "FABLE" of Johnson (1988). Branch-and-bound algorithms are also developed by Van Assche and Herroelen (1979), Hackman et al. (1989) and Nourie and Venta (1991) among others. The branch-and-bound algorithms outperform the dynamic

programming algorithms, developed by e.g. Schrage and Baker (1978) or Jackson (1956).

This paper describes an attempt to apply the branch-and-price technique to the assembly line balancing problem. In section 2, we introduce a new formulation for the SALBP-I, based on Dantzig-Wolfe decomposition and apply the Hybrid Simplex Method/Subgradient Optimization Procedure (Degraeve and Peeters 2000) to solve this formulation. In section 3, we outline a branch-and-bound algorithm to solve the pricing problem. In section 4, we discuss the initialization of the Hybrid Simplex Method/Subgradient Optimization Procedure. Section 5 presents the computational results. The computation time to solve the LP relaxation seems prohibitive to obtain competitive results with the existing branch-and-bound algorithms. Therefore, we will focus on the quality of the lower bound of the new formulation by comparing this bound with the optimal solutions for the various data sets. The development of a branch-and-price algorithm is postponed, until a faster algorithm can be developed to solve the LP relaxation.

2 New Formulation of the SALBP-I

In this Section, we present a formulation for SALBP-I, based on Dantzig-Wolfe decomposition. First, we define the earliest (e_i) and latest station (l_i) for each task i as respectively the first station to which a task can be assigned and the last station, given some upper bound m on the optimal solution. In Section 4, we discuss some heuristics to compute an upper bound m. Let P_i (F_i) be the set of tasks that immediately precede (follow) task i and let P_i^* (F_i^*) be the set of tasks which precede (follow) task i, then the earliest station and latest station for task i can be defined as follows:

$$e_i = \left\lceil (t_i + \sum_{j \in P_i^*} t_j)/c \right\rceil \tag{1}$$

$$l_i = m+1-\left\lceil t_i + \sum_{j \in F_i^*} t_j)/c \right\rceil \tag{2}$$

Let H be the set of precedence constraints, $H = \left\{(i_1, i_2) \middle| i_1 \in P_{i_2}\right\}$, N^j the set of tasks that can be assigned to station j and $H^j = \left\{(i_1, i_2) \in H : \{i_1, i_2\} \subset N^j\right\}$, then we define the set of non-forbidden tasks assignments to station j, Q_j, as follows:

$$Q_j = \{q_j \in \{0,1\}^n : \sum_{i \in N^j} t_i q_{ij} \le c \text{ and } \forall (i_1, i_2) \in H^j : \\ \forall i_3 \in \left(N^j \cap F_{i_2}^*\right): q_{i_1 j} + q_{i_3 j} - 1 \le q_{i_2 j}\} \tag{3}$$

Observe that the set of task assignments to station j contains infeasible assignments. Based on the computation of earliest and latest stations, we only know that a certain task i out of the set N^j is a *candidate* to be present in station j, but this does not assure that all task assignments belonging to Q_j are feasible for that particular station j.

Let L be a lower bound on the number of stations and let z_{q_j} be 1, if station assignment $q_j \in Q_j$ is selected for station j and 0, otherwise, than we can formulate the SALBP-I as follows:

(i) Minimize the number of stations;

$$Z^{AS1} = L + Min \sum_{j=L+1}^{m} \sum_{q_j \in Q_j} z_{q_j} \qquad \text{(AS1)} \tag{4}$$

(ii) Each task must be assigned to a station;

$$\sum_{j=e_i}^{l_i} \sum_{q_j \in Q_j} q_{ij} z_{q_j} = 1 \qquad i = 1,\ldots,n \tag{5}$$

(iii) just one station has to be chosen for each of the first L stations and maximum one for station L+1 to m;

$$\sum_{q_j \in Q_j} z_{q_j} = 1 \qquad j = 1,\ldots,L$$

$$\sum_{q_j \in Q_j} z_{q_j} \le 1 \qquad j = L+1,\ldots,m \tag{6}$$

(iv) precedence constraints;

$$\sum_{j=e_{i_2}}^{l_{i_1}} \sum_{q_j \in Q_j} j q_{i_1 j} z_{q_j} \le \sum_{j=e_{i_2}}^{l_{i_1}} \sum_{q_j \in Q} j q_{i_2 j} z_{q_j} \qquad (i_1, i_2) \in H \tag{7}$$

(v) integrality and non-negativity;

$$z_{q_j} \in \{0,1\} \qquad \forall j \in \{1,\ldots,m\}, \forall q_j \in Q_j \tag{8}$$

The LP-relaxation of this formulation can be found using column generation. Let $\pi \equiv (\pi_1,...,\pi_n)$ be the vector of dual prices of the assignment constraints (5), let $\mu \equiv (\mu_1,...,\mu_m)$ be the vector of dual prices of the station constraints (6) and let $v_{i_1 i_2}$ be the dual price of the precedence constraint corresponding to the precedence relation $(i_1,i_2)\in H$ (7). Then we state for every station j the profit vector $\overline{\pi}_j \equiv (\overline{\pi}_{1j},...,\overline{\pi}_{nj})$ of assigning task $i=1,...,n$ to station j, as follows:

$$\overline{\pi}_{ij} = \pi_i + \sum_{h\in P_i} jv_{hi} - \sum_{h\in F_i} jv_{ih} \qquad \text{for } i\in N^j$$

$$\overline{\pi}_{ij} = 0 \qquad \text{otherwise}$$

To find a new column for station j that improves the LP relaxation Z_{LP}^{AS1}, we must solve the following knapsack problem with side constraints:

$$v(\overline{\pi}_j) = Max\left\{ \sum_{i\in N^j} \overline{\pi}_{ij} x_{ij} : \sum_{i\in N^j} t_i x_{ij} \le c \text{ and } \forall (i_1,i_2)\in H^j : \right. \\ \left. \forall i_3 \in \left(N^j \cap F_{i_2}^*\right): x_{i_1 j} + x_{i_3 j} - 1 \le x_{i_2 j} \text{ and } x_{ij} \in \{0,1\} \right\} \qquad (9)$$

If the reduced cost is negative, the station improves the current LP value and it is added to the master problem:

$$\mu_j - v(\overline{\pi}_j) < 0 \qquad j=1,...,L$$

$$1+\mu_j - v(\overline{\pi}_j) < 0 \qquad j=L+1,...,m$$

The formulation AS1 (4-8) has several drawbacks. First, it has many constraints, namely $n+m+|H|$. Especially the number of precedence constraints $|H|$ can be large. The more constraints we have, the more difficult the LP-relaxation to be solved. Moreover, these constraints are rather weak and will rarely result in a better lower bound of the LP-relaxation of formulation (4-8). Secondly, the same set of tasks can be a feasible column for several stations, i.e. $q_j = q_k$, $j \neq k$. This is computationally inefficient, since it can happen that the same column is generated several times. The computational effort for solving this formulation is prohibitively high to obtain competitive results with "traditional" branch-and-bound procedures as SALOME (Scholl and Klein 1997).

Therefore, we propose a relaxation of formulation AS1 without precedence relations that is similar to formulation of Gilmore and Gomory (1961) for the CSP.

Only the set of feasible packing combinations, or station assignments differs from the BPP formulation. Let Q be the set of feasible station assignments, or

$$Q = \left\{ q \in \{0,1\}^n : \sum_{i=1}^{n} t_i q_i \le c \text{ and } \forall (i_1, i_2) \in H : \forall i_3 \in F_{i_2}^* : q_{i_1} + q_{i_3} - 1 \le q_{i_2} \right\} \tag{10}$$

Let now z_q be 1, if station assignment $q \in Q$ is selected and 0, otherwise, then we can state the relaxation of formulation AS1 for the SALBP-I as follows:

(i) Minimize the number of stations

$$Z^{AS2} = Min \sum_{q \in Q} z_q \qquad \text{(AS2)} \tag{11}$$

(ii) Each task must be assigned to a station:

$$\sum_{q \in Q} q_i z_q = 1 \qquad i = 1,\ldots,n \tag{12}$$

Since formulation *AS2* is a relaxation of formulation *AS1*, $\lceil Z^{AS2} \rceil$ is a lower bound on Z^{AS1}. Again this formulation can be solved using column generation. Observe that an integer solution of formulation (11-12) is not necessarily feasible. The precedence relations within the selected stations will not be violated, but it can happen that two stations *r* and *s* will be incompatible, because station *r* contains a predecessor of a task assigned station *s* and station *s* contains a predecessor of a task assigned to station *r*. Below in Figure 1, the stations {2,5} and {3,4,6} would be incompatible, because on one hand task 2 precedes task 4, but on the other hand, task 3 must precede task 5. The pricing problem consists of finding the column $q \in Q$ with minimal reduced cost. Let $\pi \equiv (\pi_1,\ldots,\pi_n)$ be the vector of dual prices of the assignment constraints (12) and let x_i, i=1,...,n be a binary variable that is 1, if tasks i is assigned to the station and 0, else, then we can formulate the pricing problem as follows.

(i) Minimize the reduced cost,

$$rc = Min(1 - \sum_{i=1}^{n} \pi_i x_i) = 1 - Max \sum_{i=1}^{n} \pi_i x_i \tag{13}$$

(ii) do not exceed the cycle time

$$\sum_{i=1}^{n} t_i x_i \le c \tag{14}$$

(iii) precedence relations

$$x_{i_1} + x_{i_3} - 1 \leq x_{i_2} \qquad \forall (i_1, i_2) \in H : \forall i_3 \in F^*_{i_2} \tag{15}$$

(iv) Integrality and non-negativity

$$x_i \in \{0,1\} \qquad \forall i \in N \tag{16}$$

In the next section, we outline a branch-and-bound algorithm to solve the pricing problem.

3 Branch-and-Bound Algorithm for the Pricing Problem

The pricing problem is a 0-1 knapsack problem with side constraints, namely the precedence relations (15). Although many variants of the 0-1 knapsack problem with various side constraints are discussed in the literature, there exists to the best of our knowledge, no reference in the literature to the pricing problem (13-16). Among those variants, we have e.g. the Multiple Choice Knapsack problem (Martello and Toth 1990) or the Tree Knapsack Problem (Cho and Shaw 1997). Since the 0-1 KSP is a special case of the pricing problem (13-16), the latter is *NP*-hard. We propose a branch-and-bound algorithm to solve the pricing problem (13-16) to a proven optimum. Throughout this section, we assume that the dual prices π_i, i=1,...,n, are multiplied by a scaling factor (e.g. 10000) and transformed to an integer, e.g. a dual price of 0.542 becomes 5420, which is a standard procedure to solve a knapsack problem with profit vector containing real numbers (Martello and Toth 1990). Example 1 illustrates an instance of the pricing problem.

***Example 1**: An instance of the SALBP-I*

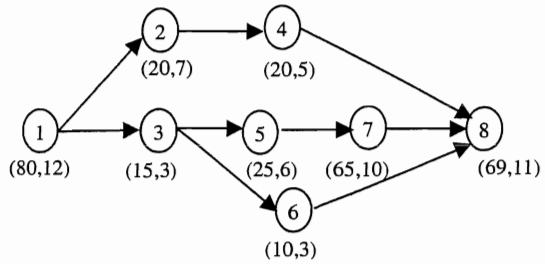
Table 1: Example SALBP-I

Task	t_i	*flw*	π_i	π_i/t_i	*srt*
1	12	2, 3	80	6.67	1
2	7	4	20	2.86	8
3	3	5, 6	15	5.00	4
4	5	8	20	4.00	6
5	6	7	25	4.17	5
6	3	8	10	3.33	7
7	10	8	65	6.50	2
8	11	-	69	6.27	3

Table 1 presents an example with 8 tasks and the cycle time c is equal to 20. The column labelled "t_i" gives the processing times of the tasks and

the column "*flw*" the tasks that immediately follow task i. The column "π_i" presents the dual prices of constraints (12). Figure 1 presents the precedence graph of the example. Below every node i, corresponding to task i, of the graph, the dual price and the processing time of task i is given, (π_i, t_i).

Figure 1: Example SALBP-I, c=20



In a pre-processing routine, before the column generation algorithm is started, we determine for every task i, i=1,...,n, the set TF_i of other tasks j that can not be assigned to the same station as task i, because the sum of the processing times of task i and j and the tasks that belong to the intersection of all the followers (predecessors) of task i and predecessors (followers) of task j, if i is smaller (greater) than j exceeds the cycle time c, or

$$TF_i = \left\{ j \in N : t_i + t_j + \sum_{k \in F_i^* \cap P_j^*} t_k + \sum_{k \in P_i^* \cap F_j^*} t_k > c \right\} \tag{17}$$

Observe that one of the two summations in expression (17) is always 0, namely the first if $j<i$, and the second if $j>i$. To explain the algorithm, we use the following notation:

$(\bar{x}_1, \bar{x}_2, ..., \bar{x}_n)$: the current solution of the problem at a node in the enumeration tree,

$\hat{c} = c - \sum_{i=1}^{n} t_i \bar{x}_i$: the remaining time,

$\hat{z} = \sum_{i=1}^{n} \pi_i \bar{x}_i$: the current objective value,

z_{best} : incumbent solution,

UB_{sp} : the upper bound on the objective value of the subproblem,

FL: forbidden list of tasks, i.e. tasks that cannot be assigned to the current station,

$\overline{FL}$: the set of items that do not belong to FL, or $\overline{FL} = N \setminus FL$.

The algorithm starts with sorting the items in decreasing order of the ratio dual price divided by the task time, i.e. π_i / t_i, i=1,...,n. Next we compute for every task i the minimum task time $t_i^{\min}$ from the set of task with ratio π_j / t_j lower than or equal to the ratio π_i / t_i, or

$$t_i^{\min} = Min\{t_j : j \in N \text{ and } \pi_j / t_j \leq \pi_i / t_i\} \tag{18}$$

We build a binary search tree, where at every level of the branch-and-bound tree a task is added to the station or not. We always first investigate the node, where the task is inserted in the station. In order to limit the search tree, the upper bound UB_{sp} is computed. This upper bound is the upper bound U_2 for the standard 0-1 knapsack problem of Martello and Toth (1990), computed over a restricted set of tasks, namely only the tasks that do not belong to the set of forbidden tasks FL. Let the set $\overline{FL}$ be equal to $\{i_1, i_2, ..., i_{\overline{n}}\}$ and assume without loss of generality that the items in $\overline{FL}$ are sorted in decreasing order of the ratio $\pi_{i_k} / t_{i_k}, k = 1,...,\overline{n}$, or $\pi_{i_1} / t_{i_1} \geq \pi_{i_2} / t_{i_2} \geq ... \geq \pi_{i_{\overline{n}}} / t_{i_{\overline{n}}}$. Next, we define the task i_h as the critical task, i.e. the task with the minimum sub-index h, for which holds that the sum of the processing times of item i_h and the items with a lower sub-index, $k<h$, exceeds the cycle time c, or

$$h = Min\left\{k \in \{1,...,\overline{n}\} : \sum_{l=1}^{k} t_{i_l} > c\right\}$$

Now, the upper bound for the subproblem UB_{sp} is equal to the maximum of $U^{(1)}$ and $U^{(2)}$, or

$$UB_{sp} = Max\{U^{(1)}, U^{(2)}\}, \tag{19}$$

where

$$U^{(1)} = \sum_{k=1}^{h-1} \pi_{i_k} + \left\lfloor (c - \sum_{k=1}^{h-1} t_{i_k}) * \pi_{i_{h+1}} / t_{i_{h+1}} \right\rfloor$$

$$U^{(2)} = \sum_{k=1}^{h-1} \pi_{i_k} + \pi_{i_h} - \left\lceil (\sum_{k=1}^{h} t_{i_k} - c) * \pi_{i_{h-1}} / t_{i_{h-1}} \right\rceil$$

The reasoning behind this bound is that for $U^{(1)}$ the critical task is not inserted in the station. After adding all tasks i_k, where $k<h$ to the station, the remaining units of the cycle time $c - \sum_{k=1}^{h-1} t_{i_k}$ are valued at their maximum possible price per time unit of the items with an higher sub-index than h, namely the ratio $\pi_{i_{h+1}} / t_{i_{h+1}}$. For $U^{(2)}$ however, the critical task is added to the station, but there is not enough cycle time left, namely we need $\sum_{k=1}^{h} t_{i_k} - c$ more units of time. The minimum price per time unit is $\pi_{i_{h-1}} / t_{i_{h-1}}$. The computation of the upper bounds will be illustrated in the example at end of this section.

We initialize the incumbent with a heuristic solution. We insert the task with the highest ratio π_i / t_i and initialize the list of available tasks, i.e. tasks that can be inserted without violating the precedence constraints (15), given the current assignments of tasks to the station. Next, we add the task on the available list with the highest ratio π_i / t_i to the station and update the list, until no task can be assigned anymore.

We distinguish four steps in the branch-and-bound procedure: (i) insertion of a task, (ii) removal of a task, (iii) updating the incumbent and (iv) backtracking.

(i) *Insertion of a task.*

The items are considered in decreasing order of the ratio dual price over task time π_i / t_i as candidates to be added to the current station. It can happen that a task can only be added to a station if some of its predecessors or followers are also added to the station. Suppose for instance that in the example of Table 1 and Figure 1, the current solution consists of task 7 and that the candidate task is task 3. In that case, task 5 must also be added to the station in order to ensure feasibility. Observe that task i precedes (or follows) the current station, if task i belongs to the union of the all predecessors (or followers) of the current solution, or

$$i \in \bigcup_{j:\bar{x}_j>0} P_j^* \quad (\text{or } i \in \bigcup_{j:\bar{x}_j>0} F_j^*)$$

Task i is parallel to the current station, if none of its predecessor or followers belongs to the current solution, or

$$i \notin \bigcup_{j:\bar{x}_j>0} \left(P_j^* \cup F_j^*\right) \text{ or } \left(P_i^* \cup F_i^*\right) \cup \{j \in N : \bar{x}_j > 0\} = \varnothing$$

In general, let task i be the candidate task, then the set of tasks TS that must be added to the current solution are

(i) $TS = \bigcup_{j:\bar{x}_j>0} P_j^* \cap F_i^* \setminus \{j \in N : \bar{x}_j > 0\}$, if task i precedes the current station, or (20a)

(ii) $TS = \bigcup_{j:\bar{x}_j>0} F_j^* \cap P_i^* \setminus \{j \in N : \bar{x}_j > 0\}$, if task i follows the current station, or (20b)

(iii) $TS = \varnothing$, if task i is parallel to the current station. (20c)

Next, we check whether task i and set TS can be inserted in the station, given the remaining cycle time, or $t_i + \sum_{j \in TS} t_j \le \hat{c}$. If so, task i and all tasks from the set TS are inserted in the station, the remaining cycle time and the current solution are updated, or

$$\hat{z} \leftarrow \hat{z} + \pi_i + \sum_{j \in TS} \pi_j \text{ and } \hat{c} \leftarrow \hat{c} - t_i - \sum_{j \in TS} t_j$$

If not, the task i and all its predecessors (followers) P_i^* (F_i^*) are added to the forbidden list, FL, if task i precedes (follows) the current station.

After adding a task i to the current station, the forbidden list of tasks is updated with the tasks of set TF_i as defined in equation (17). Moreover, if the forbidden list was not empty, before the task out of TF_i were added, we can add even more tasks to the forbidden list. Suppose in the example of Table 1 and Figure 1, that the current solution consists of task 4 and the forbidden list of task 5, or $\bar{x}_4 = 1, \bar{x}_{j \ne 4} = 0$ and $FL=\{5\}$ and that we insert item 3. In that case, tasks 7 and 8 can be added to the forbidden list. In general, the tasks that can be added to the forbidden list, given that task i is inserted in the station and task j belongs to the forbidden list, are all predecessors (followers) of task j, if task j precedes (follows) task i. The set of all tasks that can be added to the forbidden list AF, if task i is inserted in the station, is then given by the following expression:

$$AF = \bigcup_{j \in FL \cap P_i^*} P_j^* \cup \bigcup_{j \in FL \cap F_i^*} F_j^*$$

After a task is added to the current solution or to the forbidden list, the upper bound UB_{sp} is computed, using equation (19). If the upper bound UB_{sp} is equal to or smaller

than the incumbent solution z_{best}, we will remove the last inserted task as explained below in step (ii). Otherwise we proceed with the next task i, with the highest ratio π_i / t_i that is not on the forbidden list. If the remaining cycle time is greater or equal than the minimum task time $t_i^{\min}$ of equation (18), we try to insert task i, as explained above. Otherwise or if no task can be inserted anymore, because all tasks with a lower ratio are on the forbidden list, we proceed with updating the incumbent in step (iii).

(ii) *Removal of a Task from the current solution.*

If we remove task i from the current solution, we must also remove all tasks of the set TS as defined in equations (20a–20c) from the current solution. As a result, the current solution and remaining cycle time are updated as follows:

$$\hat{z} \leftarrow \hat{z} - \pi_i - \sum_{j \in TS} \pi_j \text{ and } \hat{c} \leftarrow \hat{c} + t_i + \sum_{j \in TS} t_j$$

The forbidden list FL, which is stored at each level of the branch-and-bound tree, is set equal to the forbidden list of the previous level. If the removed task i is not parallel with the current solution, the task i and all its predecessors (followers) P_i^* (F_i^*) are added to the forbidden list, FL, if task i precedes (follows) the current station.

After removing an item, we compute the upper bound UB_{sp}. If the upper bound is greater than the incumbent, we go back to step (i) and try to insert the next item. Otherwise, we backtrack as explained in point (iv).

(iii) *Update the solution*

If the current solution is greater than the incumbent, $\hat{z} > z_{best}$, we update the incumbent solution. Otherwise, we go to step (ii) and remove the last inserted item.

(iv) *Backtrack*

Backtracking consists of going to the previous level of the tree, until we find an inserted task. If there exists such task, we go to step (ii) and remove this task. If no inserted task can be found anymore, the procedure terminates and a proven optimal solution is obtained.

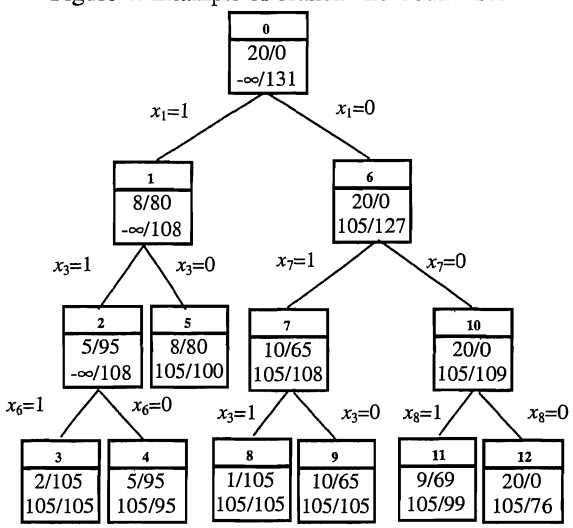
***Example 2**: Algorithm for the Pricing Problem.*

The algorithm is illustrated in Figure 2, using the example of Table 1, where the column π_i / t_i gives the ratio dual price over task time and the column "*srt*" the decreasing order of the ratio. The branch-and-bound tree consists of 13 nodes, represented by a box. In the header of the box the node number is given, while the first line of the box gives the remaining

cycle time, followed by the current solution $\hat{c}/\hat{z}$, and the second line gives the incumbent solution and the upper bound z_{best}/UB_{sp}.

In the first node, task 1, which has the highest π_i/t_i-ratio, is inserted in the station. The remaining capacity becomes 8 (=20-12) and the current solution is 80. The forbidden list is updated by the set TF_1, which consists of the tasks 4, 5, 7 and 8 and the upper bound UB_{sp} is computed over the sorted set $\overline{FL}=\{3,6,2\}$, which is equal to 80+Max\{25,28\}=108. In node 2, task 3 is inserted in the bin, and no other task must be added to ensure feasibility. In node 3, task 6 is inserted. The remaining cycle time is 2 and $t_6^{\min}=7$, as result, no tasks can be added anymore and the incumbent is updated, z_{best}=105.

Figure 4: Example of branch-and-bound tree



In node 4, task 6 is removed and no other task can be inserted, thus we backtrack. In node 5, task 3 is removed from the station. Since task 1 is still present in the bin, we can add task 6 to the forbidden list and find an upper bound of 100. As a result, node 5 can be pruned and we backtrack. In node 6, we remove task 1 from the current solution and compute the upper bound UB_{sp}. The critical item is item 8, and $UB_{sp}=\{U^{(1)},U^{(2)}\}$ =127, where

$$U^{(1)} = \pi_7 + \lfloor (c - t_7) \times \pi_3 / t_3 \rfloor = 65 + \lfloor 10 \times 5 \rfloor = 115$$

$$U^{(2)} = \pi_7 + \pi_8 - \lceil (t_7 + t_8 - c) \times \pi_8 / t_8 \rceil = 134 - 7 = 127$$

Since the UB_{sp} is greater than the best-known solution, we insert item 7 in node 7. In node 8, task 3 is the candidate to be inserted in the station, but task 7 is already present in the station. Since task 5 precedes task 7 and follows task 3, task 5 must also be inserted, which leads to a solution of 105 and a remaining cycle time of 1. Node 8 can be pruned, because UB_{sp} is equal to the incumbent solution. The same holds for node 9. As a result node 9 can be pruned and we backtrack. Nodes 10,11 and 12 complete the tree.

4 Hybrid Simplex Method/Subgradient Optimization Procedure

We solve formulation AS2 (11-12) using the hybrid simplex method/subgradient optimization procedure (Degraeve and Peeters 2000). The adaptations are straightforward. Therefore, we focus on the differences, which mainly concern the initialization of the master program.

Talbot et al. (1986) present a comparative evaluation of heuristic line balancing techniques, in which several "single pass decision rules" are discussed. The heuristic basically works as follows. For every task a "score" is computed, which reflects the task's priority to be assigned to a station. The available list is the list of tasks that for which all predecessors have been assigned to a station and for which the processing time does not exceed the remaining cycle time of the station under construction. This list is then sorted according to the priority-score and the available task with the highest score is assigned to the current station. After the assignment of a task to the station, the available list is updated. If the available list is empty, a new station is initialized, until all tasks are assigned to stations. The heuristic can also be applied in backward direction, where we start with the tasks without followers, and the tasks on the available list are those for which all followers have been assigned to a station. Talbot et al. (1986) discuss 13 single pass decision rules, that differ only in the way the score is computed. We use 3 different priority scores, ps_i, i=1,...,n namely:

(i) the processing time of the tasks (Moodie and Young 1965), $ps_i = t_i$,

(ii) the number of immediate followers (Tonge 1961), $ps_i = |F_i|$,

(iii) the minimum upper bound (Talbot et al. 1984), $ps_i = n+1-\left\lceil (t_i + \sum_{j \in F_i^*} t_j)/c \right\rceil$.

For each priority-score, we apply the heuristic in forward and backward direction. The best heuristic solution is compared with the widely used lower bound for the SALBP-I, namely the rounded up value of the sum of the task times divided by the cycle time, or

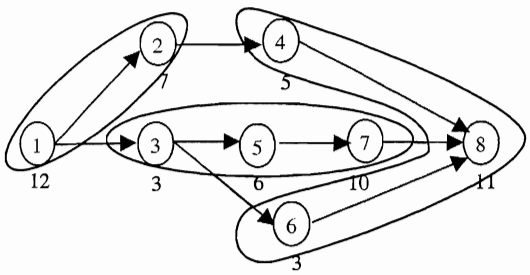
$$LB_1 = \left\lceil \sum_{i \in N} t_i / c \right\rceil \tag{21}$$

If both are equal, the procedure stops. Otherwise, formulation AS2 (11-12) is initialized with the heuristic station assignments and the hybrid simplex method/subgradient optimization procedure starts.

***Example 3**: Heuristic for the SALBP-I*

In Figure 3, we use the third priority score (the minimum upper bound) and apply the heuristic in forward direction. We find a solution of three stations, namely {1,2}, {3,5,7} and {4,6,8}, which is equal to the lower bound, $LB_1 = \lceil 57/20 \rceil$.

Figure 3: Single pass heuristic: Minimum upper bound



After initializing the Master Program, we continue in the same way as Degraeve and Peeters (2000), where we outlined the Hybrid Simplex Method/Subgradient Optimization Procedure.

5 Computational Results

In this section, we discuss the quality of the lower bound obtained by solving formulation AS2 (11-12). The experiments were run on a Dell optiplex G1 Pentium II 350 Mhz PC, using the Windows98 operating system, all computation times are given

in seconds. Our algorithm is coded in C++ and linked with the industrial LINDO optimization library version 5.3 (Schrage 1995). The procedure is tested on the same instances as the branch-and-bound procedure of Scholl and Klein (1997). They use three data sets, namely the data set of Talbot (Talbot et al. 1986), of Hoffmann (Hoffmann 1992) and Scholl (Scholl 1993). All those instance are available on http://www.bwl.tu-darmstadt.de/bwl3/, which also provides the best known solution in the literature for every instance. The data sets consist of a total of 25 precedence graphs.

Table 2: Characteristics of the precedence Graphs

Name	n	t_{min}	t_{max}	Σt_i	OS	TV
Arcus1	83	233	3691	75707	59.09	15.84
Arcus2	111	10	5689	150399	40.38	568.90
Barthold	148	3	383	5634	25.80	127.67
Barthol2	148	1	83	4234	25.80	83.00
Bowman	8	3	17	75	75.00	5.67
Buxey	29	1	25	324	50.74	25.00
Gunther	35	1	40	483	59.50	40.00
Hahn	53	40	1775	14026	83.82	44.38
Heskiaoff	28	1	108	1024	22.49	108.00
Jackson	11	1	7	46	58.18	7.00
Jaeschke	9	1	6	37	83.33	6.00
Kilbridge	45	3	55	552	44.55	18.33
Lutz1	32	100	1400	14140	83.47	14.00
Lutz2	89	1	10	485	77.55	10.00
Lutz3	89	1	74	1644	77.55	74.00
Mansoor	11	2	45	185	60.00	22.50
Mertens	7	1	6	29	52.38	6.00
Mitchell	21	1	13	105	70.95	13.00
Mukherje	94	8	171	4208	44.80	21.38
Roszieg	25	1	13	125	71.67	13.00
Sawyer	30	1	25	324	44.83	25.00
Scholl	297	5	1386	69655	58.16	277.20
Tonge	70	1	156	3510	59.42	156.00
Warnecke	58	7	53	1548	59.10	7.57
Wee-Mag	75	2	27	1499	22.67	13.50

Table 2 presents the characteristics of the precedence graphs (Scholl and Klein 1997). The first column give the name of the precedence graph, the second column the number of tasks, followed by the minimum task time (t_{min}), the maximum task time (t_{max}) and the sum of the task times (Σt_i). The last two columns gives two complexity measures, namely the Order Strength (OS) and the Time Variability Ratio (TV). The Order Strength is equal to the number of all precedence relations divided by the

maximum number of precedence relations, namely $n(n\text{-}1)/2$. Time Variability Ratio is defined as the maximum task time divided by the minimum task time, $TV=t_{max}/t_{min}$. For every precedence graph, the SALBP-I is solved for different values of the cycle time c.

Table 3a: Detailed results of data set of Talbot, part I

name	c	$UBap$	LB_1	$LBlp$	z^*	cpu	t_{LP}	mst	sp	$stat$
Bowman	20	5	4	5	5	0.00	0.00	1	0	12
Mansoor	48	4	4	4	4	0.00	0.00	0	0	10
Mansoor	62	3	3	3	3	0.00	0.00	0	0	11
Mansoor	94	2	2	2	2	0.00	0.00	0	0	4
Mertens	6	6	5	6	6	0.00	0.00	1	0	9
Mertens	7	5	5	5	5	0.00	0.00	0	0	5
Mertens	8	5	4	5	5	0.00	0.00	1	0	9
Mertens	10	3	3	3	3	0.00	0.00	0	0	5
Mertens	15	2	2	2	2	0.00	0.00	0	0	4
Mertens	18	2	2	2	2	0.00	0.00	0	0	2
Jaeschke	6	8	7	8	8	0.00	0.00	1	0	11
Jaeschke	7	7	6	7	7	0.00	0.00	1	0	10
Jaeschke	8	6	5	6	6	0.00	0.00	1	0	10
Jaeschke	10	4	4	4	4	0.00	0.00	0	0	4
Jaeschke	18	3	3	3	3	0.00	0.00	0	0	3
Jackson	7	8	7	8	8	0.00	0.00	1	0	15
Jackson	9	6	6	6	6	0.00	0.00	0	0	6
Jackson	10	6	5	5	5	0.00	0.00	1	4	21
Jackson	13	4	4	4	4	0.00	0.00	0	0	4
Jackson	14	4	4	4	4	0.00	0.00	0	0	4
Jackson	21	3	3	3	3	0.00	0.00	0	0	3
Mitchell	14	8	8	8	8	0.00	0.00	0	0	8
Mitchell	15	8	7	8	8	0.05	0.00	2	69	44
Mitchell	21	5	5	5	5	0.00	0.00	0	0	13
Mitchell	26	5	5	5	5	0.00	0.00	0	0	5
Mitchell	35	3	3	3	3	0.00	0.00	0	0	9
Mitchell	39	3	3	3	3	0.00	0.00	0	0	3
Heskiaoff	138	8	8	8	8	0.00	0.00	0	0	8
Heskiaoff	205	5	5	5	5	0.00	0.00	0	0	18
Heskiaoff	216	5	5	5	5	0.00	0.00	0	0	0
Heskiaoff	256	4	4	4	4	0.00	0.00	0	0	15
Heskiaoff	324	4	4	4	4	0.00	0.00	0	0	4
Heskiaoff	342	3	3	3	3	0.00	0.00	0	0	9
Sawyer	25	14	13	14	14	0.11	0.00	4	180	90
Sawyer	27	14	12	13	13	0.11	0.00	3	150	80
Sawyer	30	12	11	12	12	0.11	0.00	4	209	97
Sawyer	36	10	9	10	10	0.16	0.05	4	205	105
Sawyer	41	8	8	8	8	0.00	0.00	0	0	27
Sawyer	54	7	6	7	7	0.11	0.00	4	176	98
Sawyer	75	5	5	5	5	0.00	0.00	0	0	5

Table 3a and Table 3b present the results of the 64 instances of the data set of Talbot. The column headings have the following meaning:

Name : Name of the precedence graph,

c : The cycle time,

UB_{ap} : Best upper bound found by the heuristics used to initialized the Master Program,

LB_1 : The lower bound of equation (22),

$LB_{lp}=\lceil Z^{AS2} \rceil$: Lower bound of the formulation (11-12)

z^* : Best known solution in the literature.

cpu : Time needed to solve formulation (11-12) in seconds

t_{LP} : Time needed to solve the master LP's using the simplex method.

mst : The number of solved master problems.

sp : The number solved of pricing problems.

stat : The number of stations generated (number of columns in the Master).

Table 3b: Detailed results of data set of Talbot, part II

name	*c*	*UBap*	LB_1	*LBlp*	*z**	*cpu*	t_{LP}	*mst*	*sp*	*stat*
Kilbridge	57	10	10	10	10	0.06	0.00	0	0	10
Kilbridge	79	8	7	7	7	0.27	0.11	2	180	149
Kilbridge	92	7	6	6	6	0.44	0.16	3	270	194
Kilbridge	110	6	6	6	6	0.00	0.00	0	0	6
Kilbridge	138	5	4	4	4	0.66	0.27	3	234	188
Kilbridge	184	3	3	3	3	0.00	0.00	0	0	9
Tonge	176	21	20	20	21	0.77	0.16	2	280	225
Tonge	364	10	10	10	10	0.00	0.00	0	0	10
Tonge	410	9	9	9	9	0.00	0.00	0	0	9
Tonge	468	8	8	8	8	0.00	0.00	0	0	8
Tonge	527	7	7	7	7	0.06	0.00	0	0	7
Arcus1	5048	16	15	16	16	1.92	0.60	4	489	332
Arcus1	5853	14	13	14	14	2.30	0.93	4	500	347
Arcus1	6842	12	12	12	12	0.00	0.00	0	0	12
Arcus1	7571	11	10	11	11	17.91	8.33	17	1047	519
Arcus1	8412	10	9	10	10	2.97	1.48	4	503	358
Arcus1	8898	9	9	9	9	0.00	0.00	0	0	9
Arcus1	10816	8	7	8	8	19.99	10.39	14	1329	603
Arcus2	5755	27	27	27	27	0.00	0.00	0	0	57
Arcus2	8847	18	17	18	18	16.53	5.28	10	748	539
Arcus2	10027	16	15	16	16	18.34	5.12	7	906	578
Arcus2	10743	15	14	15	15	15.82	5.72	6	589	474
Arcus2	11378	14	14	14	14	0.00	0.00	0	0	14
Arcus2	17067	9	9	9	9	0.00	0.00	0	0	9

Table 4 presents the detailed results for the Hoffmann data set and Tables 5a to 5c for the Scholl data set. Four instances are open, i.e. a proven optimum has not yet been obtained in the literature, namely, Arcus2 c=7520, Wee-Mag c=47, Scholl c=1483 and Scholl c=1699. The lower bound of formulation AS2 (11-12) shows that the best known solution of 21 for the instance Arcus2 c=7520, is the optimal solution. Concerning the data set of Scholl, we remark that our column generation procedure cannot find a lower bound for the instance Scholl c=2680, since LINDO failed to

optimize the LPs in reasonable time, using the simplex method due to numerical problems.

Table 4: Detailed results of data set of Hoffmann

name	c	UBap	LB_1	LBlp	z*	cpu	t_{LP}	mst	sp	stat
Sawyer	27	14	12	13	13	0.11	0.11	3	150	80
Sawyer	30	12	11	12	12	0.11	0.00	4	209	97
Sawyer	33	11	10	10	11	0.05	0.05	2	120	83
Sawyer	36	10	9	10	10	0.16	0.00	4	205	105
Sawyer	41	8	8	8	8	0.00	0.00	0	0	27
Sawyer	47	8	7	7	7	0.11	0.11	2	120	94
Sawyer	54	7	6	7	7	0.11	0.06	4	176	98
Kilbridge	56	11	10	10	10	0.11	0.05	1	90	105
Kilbridge	62	10	9	9	9	0.22	0.10	2	180	148
Kilbridge	69	9	8	8	8	0.28	0.06	2	180	165
Kilbridge	79	8	7	7	7	0.28	0.11	2	180	149
Kilbridge	92	7	6	6	6	0.44	0.17	3	270	194
Kilbridge	111	6	5	5	5	0.38	0.16	2	180	151
Tonge	160	23	22	23	23	1.59	0.33	8	575	282
Tonge	168	22	21	22	22	1.65	0.27	5	560	270
Tonge	176	21	20	20	21	0.77	0.17	2	280	225
Tonge	185	20	19	19	20	1.42	0.32	4	496	269
Tonge	195	19	18	19	19	1.38	0.39	9	453	298
Tonge	207	18	17	17	18	0.77	0.34	2	280	243
Tonge	220	17	16	16	17	0.88	0.33	2	280	236
Tonge	234	16	15	15	16	1.04	0.38	3	347	263
Tonge	251	15	14	14	14	0.77	0.32	2	280	242
Tonge	270	14	13	13	14	1.21	0.50	3	420	281
Tonge	293	13	12	12	13	0.88	0.44	2	280	232
Tonge	320	12	11	11	11	0.77	0.39	2	234	237
Arcus1	3786	22	20	21	21	1.05	0.56	3	345	297
Arcus1	3985	21	19	20	20	1.10	0.43	2	314	273
Arcus1	4206	20	18	19	19	0.98	0.37	3	342	284
Arcus1	4454	18	17	18	18	1.32	0.49	3	369	297
Arcus1	4732	17	16	17	17	1.26	0.59	3	343	288
Arcus1	5048	16	15	16	16	1.86	0.59	4	489	332
Arcus1	5408	15	14	15	15	2.63	1.20	4	499	358
Arcus1	5824	14	13	14	14	1.87	0.60	3	465	338
Arcus1	6309	13	12	13	13	2.20	1.11	4	504	350
Arcus1	6883	12	11	12	12	1.98	1.22	3	347	289
Arcus1	7571	11	10	11	11	17.63	8.31	17	1047	519
Arcus2	5785	27	26	27	27	7.03	2.96	4	668	477
Arcus2	6016	26	25	26	26	3.46	1.97	3	473	420
Arcus2	6267	25	24	25	25	29.11	3.03	5	793	520
Arcus2	6540	24	23	24	24	8.24	2.24	3	605	490
Arcus2	6837	23	22	23	23	21.25	3.52	11	929	590
Arcus2	7162	22	21	22	22	6.21	2.60	3	505	455
Arcus2	7520	21	20	21	21	9.99	3.60	8	695	540
Arcus2	7916	20	19	20	20	16.26	4.68	8	1114	531
Arcus2	8356	19	18	19	19	10.11	5.05	4	671	518
Arcus2	8847	18	17	18	18	16.53	5.34	10	748	539
Arcus2	9400	17	16	17	17	15.87	4.56	4	738	552
Arcus2	10027	16	15	16	16	17.96	5.00	7	906	578
Arcus2	10743	15	14	15	15	15.66	5.66	6	589	474
Arcus2	11570	14	13	13	13	22.19	5.34	4	871	616

Table 5a: Detailed results of data set of Scholl, part I

name	c	UBap	LB_1	LBlp	z*	cpu	t_{LP}	mst	sp	stat
Roszieg	14	10	9	10	10	0.00	0.00	2	69	53
Roszieg	16	9	8	8	8	0.00	0.00	2	100	59
Roszieg	18	8	7	8	8	0.06	0.00	4	150	70
Roszieg	21	6	6	6	6	0.00	0.00	0	0	8
Roszieg	25	6	5	6	6	0.06	0.00	5	172	67
Roszieg	32	4	4	4	4	0.00	0.00	0	0	6
Buxey	27	13	12	13	13	0.05	0.00	1	47	66
Buxey	30	12	11	12	12	0.11	0.00	4	175	78
Buxey	33	11	10	11	11	0.10	0.00	4	195	91
Buxey	36	10	9	10	10	0.11	0.05	3	159	87
Buxey	41	8	8	8	8	0.00	0.00	0	0	19
Buxey	47	8	7	7	7	0.06	0.06	2	116	90
Buxey	54	7	6	7	7	0.11	0.06	4	175	100
Lutz1	1414	11	10	11	11	0.06	0.00	2	91	64
Lutz1	1572	10	9	10	10	0.05	0.00	2	72	59
Lutz1	1768	9	8	9	9	0.06	0.00	3	142	68
Lutz1	2020	8	7	8	8	0.11	0.00	3	143	76
Lutz1	2357	7	6	7	7	0.11	0.11	3	174	81
Lutz1	2828	6	5	6	6	0.11	0.00	3	164	74
Gunther	41	15	12	14	14	0.00	0.00	2	98	74
Gunther	44	12	11	12	12	0.11	0.00	4	209	95
Gunther	49	11	10	10	11	0.16	0.05	3	191	108
Gunther	54	10	9	9	9	0.11	0.05	2	140	83
Gunther	61	9	8	9	9	0.17	0.06	4	227	102
Gunther	69	8	7	8	8	0.27	0.00	4	220	125
Gunther	81	7	6	7	7	0.39	0.00	5	280	137
Hahn	2004	8	7	8	8	0.16	0.05	2	113	71
Hahn	2338	7	6	7	7	0.50	0.12	3	232	111
Hahn	2806	6	5	6	6	2.03	0.06	7	537	158
Hahn	3507	5	4	5	5	0.33	0.10	3	214	63
Hahn	4676	4	3	4	4	0.06	0.00	1	104	40
Warnecke	54	33	29	31	31	0.39	0.00	3	232	140
Warnecke	56	33	28	29	29	0.27	0.05	2	232	142
Warnecke	58	30	27	29	29	0.11	0.00	2	116	124
Warnecke	60	29	26	27	27	0.33	0.06	3	348	182
Warnecke	62	29	25	27	27	0.28	0.05	3	232	138
Warnecke	65	27	24	25	25	0.33	0.11	3	348	165
Warnecke	68	25	23	24	24	0.33	0.17	4	348	188
Warnecke	71	25	22	23	23	0.22	0.00	3	235	178
Warnecke	74	23	21	22	22	0.33	0.00	3	247	177
Warnecke	78	22	20	21	21	1.10	0.12	6	580	225
Warnecke	82	21	19	20	20	0.44	0.12	3	274	196
Warnecke	86	20	18	19	19	0.49	0.15	4	369	219
Warnecke	92	19	17	17	17	0.38	0.10	2	232	179
Warnecke	97	17	16	17	17	0.82	0.11	5	464	222
Warnecke	104	16	15	15	15	0.33	0.05	2	232	206
Warnecke	111	15	14	14	14	0.44	0.05	3	347	214
Wee-mag	28	63	54	63	63	0.11	0.00	2	150	101
Wee-mag	29	63	52	63	63	0.06	0.00	1	0	103
Wee-mag	30	63	50	62	62	0.16	0.00	2	150	103
Wee-mag	31	62	49	62	62	0.11	0.00	2	150	109
Wee-mag	32	61	47	61	61	0.06	0.00	1	0	105
Wee-mag	33	61	46	61	61	0.00	0.00	1	0	103
Wee-mag	34	61	45	61	61	0.00	0.00	1	0	104
Wee-mag	35	60	43	60	60	0.00	0.00	1	0	105
Wee-mag	36	60	42	60	60	0.06	0.00	1	0	105
Wee-mag	37	60	41	60	60	0.00	0.00	1	0	108
Wee-mag	38	60	40	60	60	0.00	0.00	1	0	104

Table 5b: Detailed results of data set of Scholl, part II

name	c	UBap	LB_1	LBlp	z*	cpu	4	mst	sp	stat
Wee-mag	39	60	39	60	60	0.00	0.00	1	0	101
Wee-mag	40	60	38	60	60	0.06	0.00	1	0	98
Wee-mag	41	59	37	59	59	0.00	0.00	1	0	103
Wee-mag	42	55	36	55	55	0.05	0.00	1	0	121
Wee-mag	43	50	35	50	50	0.00	0.00	1	0	128
Wee-mag	45	39	34	38	38	1.26	0.00	3	300	180
Wee-mag	46	36	33	34	34	0.82	0.00	2	150	193
Wee-mag	47	34	32	32	33	4.45	0.17	2	300	233
Wee-mag	49	32	31	32	32	8.35	0.05	3	300	187
Wee-mag	50	32	30	32	32	3.74	0.00	2	150	203
Wee-mag	52	31	29	31	31	4.83	0.10	1	149	177
Wee-mag	54	31	28	31	31	4.67	0.12	2	150	166
Wee-mag	56	31	27	30	30	0.11	0.06	1	0	158
Lutz2	11	51	45	48	49	0.77	0.06	5	712	208
Lutz2	12	47	41	44	44	0.55	0.17	4	534	206
Lutz2	13	42	38	39	40	0.44	0.11	3	426	237
Lutz2	14	40	35	36	37	0.44	0.22	3	357	245
Lutz2	15	35	33	33	34	0.22	0.11	1	178	213
Lutz2	16	34	31	31	31	0.27	0.10	1	178	197
Lutz2	17	31	29	29	29	0.28	0.12	1	178	206
Lutz2	18	29	27	28	28	0.88	0.33	4	618	280
Lutz2	19	27	26	26	26	0.61	0.29	2	356	247
Lutz2	20	26	25	25	25	0.33	0.16	1	178	202
Lutz2	21	25	24	24	24	0.33	0.11	1	178	212
Lutz3	75	24	22	23	23	1.75	0.33	5	753	308
Lutz3	79	23	21	22	22	4.17	0.59	10	1023	336
Lutz3	83	22	20	21	21	2.63	0.98	6	895	338
Lutz3	87	21	19	20	20	1.27	0.49	4	539	313
Lutz3	92	19	18	18	19	1.59	0.54	3	534	307
Lutz3	97	18	17	18	18	3.62	0.72	6	920	373
Lutz3	103	17	16	17	17	4.23	0.82	8	1246	362
Lutz3	110	16	15	15	15	2.36	0.99	4	712	369
Lutz3	118	15	14	14	14	1.43	0.60	3	534	316
Lutz3	127	14	13	14	14	4.72	1.26	8	1107	390
Lutz3	137	13	12	13	13	3.30	1.33	6	918	391
Lutz3	150	12	11	11	12	1.70	0.70	3	534	314
Mukherje	176	25	24	25	25	1.70	1.21	3	423	328
Mukherje	183	24	23	24	24	0.50	0.39	1	123	189
Mukherje	192	23	22	23	23	0.49	0.44	1	93	178
Mukherje	201	23	21	22	22	0.49	0.33	1	144	204
Mukherje	211	21	20	21	21	0.83	0.55	1	162	200
Mukherje	222	21	19	20	20	1.59	1.16	3	377	291
Mukherje	234	19	18	19	19	1.32	0.89	2	278	263
Mukherje	248	18	17	18	18	2.97	1.61	6	940	295
Mukherje	263	17	16	17	17	1.21	0.65	3	376	301
Mukherje	281	16	15	16	16	1.70	1.05	2	292	255
Mukherje	301	15	14	15	15	2.75	1.31	4	564	294
Mukherje	324	14	13	14	14	2.36	1.32	4	455	328
Mukherje	351	13	12	13	13	2.47	1.31	3	378	286
Barthold	403	15	14	14	14	10.22	6.92	2	592	464
Barthold	434	14	13	13	13	9.89	6.54	2	592	473
Barthold	470	13	12	12	12	8.95	5.93	2	487	430
Barthold	513	12	11	11	11	11.64	7.29	2	592	477
Barthold	564	11	10	10	10	15.54	8.13	3	683	550
Barthold	626	10	9	9	9	19.17	10.16	3	660	558
Barthold	705	9	8	8	8	23.18	10.05	3	800	673
Barthold	805	8	7	7	7	42.02	19.22	4	724	624

Table 5c: Detailed results of data set of Scholl, part III

name	c	UBap	LB_1	LBlp	z*	cpu	t_{LP}	mst	sp	stat
Barthol2	84	54	51	51	51	1.65	1.16	1	296	435
Barthol2	85	53	50	50	50	2.58	1.70	2	592	546
Barthol2	87	51	49	49	49	3.08	2.15	2	592	565
Barthol2	89	50	48	48	48	2.96	1.98	2	592	554
Barthol2	91	49	47	47	47	1.92	1.48	1	296	446
Barthol2	93	48	46	46	46	1.81	1.31	1	296	447
Barthol2	95	47	45	45	45	2.03	1.54	1	296	440
Barthol2	97	46	44	44	44	2.14	1.70	1	296	428
Barthol2	99	45	43	43	43	3.02	2.09	2	592	625
Barthol2	101	44	42	42	42	3.08	2.09	2	592	597
Barthol2	104	43	41	41	41	2.09	1.59	1	296	431
Barthol2	106	42	40	40	40	2.75	1.93	2	592	605
Barthol2	109	41	39	39	39	2.75	1.87	2	592	594
Barthol2	112	39	38	38	38	4.67	3.63	2	592	578
Barthol2	115	39	37	37	37	2.25	1.75	1	296	433
Barthol2	118	38	36	36	36	3.35	2.31	2	592	595
Barthol2	121	36	35	35	35	3.30	2.42	2	592	597
Barthol2	125	35	34	34	34	3.84	2.85	2	592	588
Barthol2	129	34	33	33	33	3.85	2.91	2	592	563
Barthol2	133	33	32	32	32	3.35	2.25	2	592	564
Barthol2	137	32	31	31	31	3.85	2.80	2	592	568
Barthol2	142	31	30	30	30	4.61	3.52	2	592	563
Barthol2	146	30	29	29	29	3.74	2.58	2	592	571
Barthol2	152	29	28	28	28	3.90	2.87	2	517	513
Barthol2	157	28	27	27	27	4.95	3.79	2	592	553
Barthol2	163	27	26	26	26	5.88	4.30	2	592	535
Barthol2	170	26	25	25	25	4.73	3.42	2	592	533
Scholl	1394	52	50	50	50	66.57	48.83	2	1188	1054
Scholl	1422	51	49	50	50	107.38	54.42	5	2376	1333
Scholl	1452	50	48	48	48	49.15	31.30	2	1188	1040
Scholl	1483	49	47	47	48	126.66	92.94	3	1782	1214
Scholl	1515	47	46	46	46	50.64	31.96	2	1188	1060
Scholl	1548	47	45	46	46	82.61	41.42	5	1786	1254
Scholl	1584	46	44	44	44	67.67	50.68	2	1188	1028
Scholl	1620	44	43	43	44	80.31	50.48	4	1494	1254
Scholl	1659	43	42	42	42	91.40	56.19	3	1782	1308
Scholl	1699	42	41	41	42	236.46	198.80	3	1782	1251
Scholl	1742	42	40	40	40	59.76	39.17	2	1188	1026
Scholl	1787	41	39	39	39	58.50	35.25	2	1188	1031
Scholl	1834	39	38	38	38	73.66	50.04	2	1188	1073
Scholl	1883	38	37	37	37	95.30	55.42	3	1689	1257
Scholl	1935	37	36	36	36	148.84	105.78	3	1782	1277
Scholl	1991	36	35	35	35	60.97	35.65	2	1188	976
Scholl	2049	35	34	34	34	125.56	66.91	3	1782	1226
Scholl	2111	34	33	33	33	131.44	83.00	3	1782	1307
Scholl	2177	33	32	32	32	113.59	69.04	3	1508	1243
Scholl	2247	32	31	31	31	169.94	115.49	3	1782	1212
Scholl	2322	31	30	30	30	115.95	64.74	3	1782	1258
Scholl	2402	30	29	29	29	149.84	97.05	3	1628	1281
Scholl	2488	29	28	28	28	170.70	112.15	3	1782	1385
Scholl	2580	28	27	27	27	152.75	93.92	3	1675	1275
Scholl	2680	27	26	-	26	-	-	-	-	-
Scholl	2787	26	25	25	25	177.46	113.63	3	1618	1299

The computational results for the three data sets are summarized in Table 6. The first column gives the name of the data set and the second column labelled "*np*" the

number of instances in the data sets. Columns 3 to 6 present respectively the number of times that the best upper bound is equal (UB_{ap}) to LB_1 of equation (21), that LB_1 is equal to the best known optimal solution z^*, that the LP-bound of formulation AS2 (11-12) is equal to the optimal solution and finally that the best upper bound is equal to the optimal solution. The number in brackets denotes the fraction of the instances of the data set for which the presented criterion holds. Column 7 and 8 give the average cpu-time is and the average time that the procedure must solve the master LP's using the simplex method, and the fraction of the total time in brackets.

Table 6: Summary of the computational tests

Data Set	*np*	$UB_{ap}=LB_1$	$LB_1=z^*$	$LB_{lp}=z^*$	$UB_{ap}=z^*$	*cpu*	t_{LP}
Talbot	64	38 (.59)	42 (.66)	63 (.98)	59 (.92)	1.54	0.60 (.39)
Hoffm.	50	1 (.02)	11 (.22)	42 (.84)	36 (.72)	4.98	1.52 (.31)
Scholl	167	3 (.02)	71 (.43)	156 (.93)	66 (.40)	18.51	11.73 (.63)

From Table 6 we draw the following conclusions. (i) The data set of Talbot is very easy to solve, for 59% of the instances, a proven optimum is obtained by applying the simple heuristics to initialize the master and comparing them to the lower bound LB_1. The performance of the heuristics is even better, in 92% of the cases they are able to find a solution equal to the optimal solution. The LP-based lower bound, LB_{lp} is equal to the optimal solution for 98% of the instances for the Talbot data set. (ii) By applying the simple heuristics and computing LB_1, we can rarely solve the instances of Hoffmann and Scholl. The quality of the lower bound LB_{lp} however is still good. For respectively 84% and 93%, it is equal to the best known solution in the literature. (iii) Optimizing the master LPs with the simplex method is very time consuming. For the Scholl data set for instances, 63% of the total time to compute the LP lower bound is spent in optimizing the Master LPs using the simplex method. As mentioned above, one instance (Scholl c=2680) could not be solved due to numerical problems of the LP solver.

The master LPs to be solved are the LP relaxation of Set Covering Problems (SCP). In the literature, we find references that those LP problems are indeed hard and can not be solved efficiently by the Simplex Method. Etcheberry (1977) was the first to use subgradient optimization instead of the simplex method to solve the LP-relaxation of set covering problems. He reports: "Computational experience shows that these bounds [computed by subgradient optimization] are at least one order of

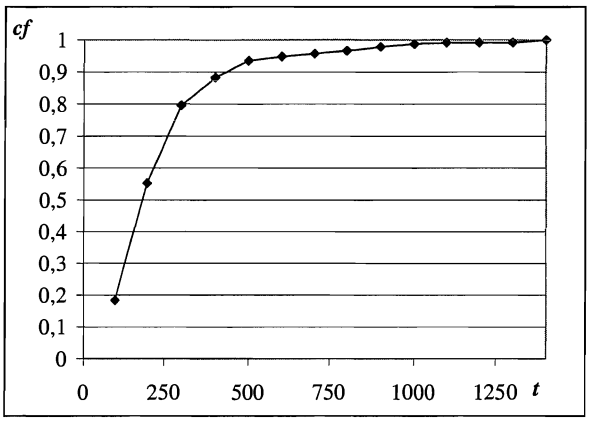
magnitude more efficient than the ones obtained by solving the associated linear program with the simplex method." Balas and Ho (1980) develop a set covering algorithm using cutting planes, heuristics and subgradient optimization. Balas and Carrera (1996) present a dynamic subgradient-based branch-and-bound procedure for set covering, explicitly mentioning that "main advantage of subgradient optimization over the simplex method (...) is its low computational cost".

In order to assess the potential of the new LP-based bound of formulation (11-12), we discuss the optimal branch-and-bound algorithms. Scholl and Klein (1999) compare four branch-and-bound methods, namely EUREKA of Hoffmann (1992), FABLE of Johnson (1988), OptPack of Nourie and Venta (1991) and SALOME of Scholl and Klein (1997). Their experiments were run on an IBM compatible personal computer 80486 with DX2-66 central processing unit, which is much slower than our PC, roughly a factor 30 to 40. They conclude that SALOME is the best procedure, especially on the more challenging Scholl data set. They also present an improved version of SALOME that is able to solve 96% of the instances of the Hoffman data set within 500 sec. The average cpu time is 23.3 seconds, including the unsolved instances. Concerning the Scholl data sets, the procedure is able to solve 87% of the instances within 500 sec and average cpu time of 82.9 seconds. Given the fact that our PC is much faster than the DX2-66, the average time to find the LP lower bound is higher than their average time to find a proven optimal solution.

However, these results should be interpreted with care. First of all, only 25 different graphs have been used in the experiments, of which 10 graphs consist of not more than 30 tasks (see Table 2). It can be expected that the nature of the precedence graph has an important impact on the computation time. Secondly, the intervals $[1,f^*c]$ in which the item weights are uniformly distributed, and the number of different items, have a large impact on the computation time in the case of bin packing instances (e.g. Degraeve and Peeters, 2002). The upper limit f^*c is expressed as a fraction f of the bin capacity c. For the SALBP, only 25 different instances with a given number of tasks and given task times are used. Only the cycle time, equivalent to the bin capacity in the BPP, varies and as a result the fraction that determines the upper limit of the interval, $f=t_{max}/c$. For the Scholl precedence graph for instances, the maximum task time is 1386 (see Table 2) and the cycle time varies between 1394 and 2787 (see Table 5c). Consequently, the fraction f varies between 0.5 and 1. It seems that indeed the branch-and-bound procedures are tested for an appropriate set of

intervals. However, a closer look at the cumulative frequency of the tasks times presented in Figure 4 for the Scholl precedence graph, learns us that about 93% of the tasks times is smaller than 500. As a result, the task times are rather distributed in the interval $[1,f^*c]$, where f varies between 0.18 and 0.36 plus some larger tasks times than where f varies between 0.5 and 1.

Figure 4: Distribution of the task times of the Scholl precedence graph



6 Conclusions and Future Research

We have investigated the quality of the LP lower bound $\lceil Z^{AS2} \rceil$. For the majority of the instances, the bound is equal to the optimal solution. However, it requires long computation times, especially when the number of tasks is high. As a result, it seems to be impossible to obtain competitive results with the existing branch-and-bound algorithms using branch-and-price, at least on the data sets currently used in the literature.

In the discussion of the computational results, we have stressed that the computational results of Scholl and Klein (1999) should be interpreted with care, because it is only tested on a very limited set of precedence graphs. A possible research direction is the validation of their algorithm on a larger set of problem instances. If it turns out that the procedure of Scholl and Klein (1997) is not able to solve certain classes of the instances, our column generation procedure can possibly be helpful to find a good lower bound and the development of a branch-and-price algorithm can be taken into consideration. An even more interesting research direction

could be another method to solve the LP formulation AS2 (11-12). Most performing procedures to solve the set covering problem (Balas and Carrera 1996) do not use the simplex method. Since the AS2 is a set covering problem, the question arises whether we should we not give up the simplex method and only rely on other techniques to find an approximation of the LP-bound.

References:

BALAS, E. AND A. HO. 1980. Set Covering Algorithms using Cutting Planes, Heuristics and Subgradient Optimization: A Computational Study. *Mathematical Programming* 12, 37-60.

BALAS, E. AND M.C. CARRERA. 1996. A Dynamic Subgradient-Based Branch-and-Bound Procedure for Set Covering. *Operations research* 44, 875-888.

BAYBARS, I. 1986. A Survey of Exact Algorithms for the Simple Assembly Line Balancing Problem. *Management Science* 32, 909-932.

CHO, C. AND D.X. SHAW. 1997. A Depth-First Dynamic Programming Algorithm for the Tree Knapsack Problem. *INFORMS Journal on Computing* 9, 431-438.

DEGRAEVE, Z. AND M. PEETERS. 2000. Solving the Linear Programming Relaxation of Cutting and Packing Problems: A Hybrid Simplex Method/Subgradient Optimization Procedure. Onderzoeksrapport NR 0017, Departement Toegepaste Economische Wetenschappen, K.U.Leuven.

DEGRAEVE, Z. AND M. PEETERS. 2002. Optimal Integer Solutions to Industrial Cutting Stock Problems: Part 2, Benchmark Results. *INFORMS Journal on Computing, accepted for publication.*

ETCHEBERRY, J. 1977. The Set Covering Problem: A new implicit enumeration algorithm. *Operations Research* 25, 760-772.

GILMORE, P.C. AND R.E. GOMORY. 1961. A Linear Programming Approach to the Cutting Stock Problem. *Operations Research* 9, 849-859.

HACKMAN, S.T., M.J. MAGAZINE AND T.S. WEE. 1989. Fast, effective algorithms for Simple Assembly Line Balancing Problems. *Operations Research* 37, 916-924.

HOFFMAN, T.R. 1992. EUREKA, an Hybrid System for Assembly Line Balancing. *Management Science* 38, 39-47.

JACKSON, J.R. 1956. A Computing Procedure for a Line Balancing Problem. *Management Science* 2, 261-271.

JOHNSON, R.V. 1988. Optimally Balancing Large Assembly Lines with 'FABLE'. *Management Science* 34, 240-253.

MARTELLO, S. AND P. TOTH. 1990. *Knapsack Problems : Algorithms and Computer Implementations.* Wiley Interscience Series in Discrete Mathematics and Optimization, John Wiley & Sons, Chichester.

MOODIE, C.L. AND H.H. YOUNG. 1965. A Heuristic Method of Assembly Line Balancing for Assumptions of Constant or Variable Work Element Times. Journal of Industrial Engineering, 16.

NOURIE, F.J. AND E.R. VENTA. 1991. Finding Optimal Line Balances with OptPack. *Operations Research Letters* 10, 165-171.

SCHOLL, A. 1993. Data of Assembly Line Balancing Problems. Working Paper, TU Darmstadt.

SCHOLL, A. AND R. KLEIN. 1997. SALOME: A Bidirectional Branch-and-Bound Procedure for Assembly Line Balancing. *INFORMS Journal on Computing* 9, 319-334.

SCHOLL, A. AND R. KLEIN. 1999. Balancing assembly lines effectively – A computational comparison. *European Journal on Operational Research* 114, 50-58.

SCHRAGE, L. 1995. LINDO : Optimization Software for Linear Programming. Lindo Systems Inc., Chicago, IL.

SCHRAGE, L AND K.R. BAKER. 1978. Dynamic Programming Solutions of Sequencing Problems with Precedence Constraints. *Operations Research* 26, 444-449

TALBOT, F.B. AND J.H. PATTERSON. 1984. An Integer Programming Algorithm with Network Cuts for Solving the Assembly Line Balancing Problem. *Management Science* 30, 85-99.

TALBOT, F.B., J.H. PATTERSON AND W.V. GEHRLEIN. 1986. A Comparative Evaluation of Heuristic Line Balancing Techniques. *Management Science* 32, 430-454.

TONGE, F.M. 1961. *A Heuristic Program of Assembly Line Balancing.* Prentice-Hall, Englewood Cliffs, N.J.

VAN ASSCHE, F. AND W. HERROELEN. 1979. An Optimal Procedure for the Single-Model Deterministic Assembly Line Balancing Problem. *European Journal of Operational Research* 3, 142-149.